



Worksheet 3 Lists and linked lists **Answers**

Task 1

1. 'Random Clothing Task' - Complete the following to show the operations implemented on a list of clothing items, initialised as an empty list `clothes[]`

Operation	List	Returns
<code>isEmpty()</code>	<code>[]</code>	<code>True</code>
<code>len()</code>	<code>[]</code>	<code>0</code>
<code>append("socks")</code>	<code>["socks"]</code>	
<code>append("shoes")</code>	<code>["socks", "shoes"]</code>	
<code>append("hat")</code>	<code>["socks", "shoes", "hat"]</code>	
<code>append("socks")</code>	<code>["socks", "shoes", "hat", "socks"]</code>	
<code>count("socks")</code>	<code>["socks", "shoes", "hat", "socks"]</code>	<code>2</code>
<code>index("shoes")</code>	<code>["socks", "shoes", "hat", "socks"]</code>	<code>1</code>
<code>len(clothes)</code>	<code>["socks", "shoes", "hat", "socks"]</code>	<code>4</code>
<code>insert(2, "gloves")</code>	<code>["socks", "shoes", "gloves", "hat", "socks"]</code>	
<code>remove("socks")</code>	<code>["shoes", "gloves", "hat", "socks"]</code>	<code>"socks"</code>
<code>pop()</code>	<code>["shoes", "gloves", "hat"]</code>	<code>"socks"</code>
<code>remove("shirt")</code>	<code>["shoes", "gloves", "hat"]</code>	<code>Error</code>
<code>append("socks")</code>	<code>["shoes", "gloves", "hat", "socks"]</code>	
<code>append("shorts")</code>	<code>["shoes", "gloves", "hat", "socks", "shorts"]</code>	
<code>len(clothes)</code>	<code>["shoes", "gloves", "hat", "socks", "shorts"]</code>	<code>5</code>
<code>index("gloves")</code>	<code>["shoes", "gloves", "hat", "socks", "shorts"]</code>	<code>1</code>
<code>pop(1)</code>	<code>["shoes", "hat", "socks", "shorts"]</code>	<code>"gloves"</code>



Task 2

2. An unsorted list contains integers in the range 0-150. The following pseudocode has been written to count and print the number of integers that are in the range 80-100, and then to remove these numbers from the list and print the amended list.

```
list1 = [34,56,34,26,80,57,98,100,80,64,102,300,35,6,87,88]
count = 0
for index = 0 to (len(list1) - 1)
    if (list1[index] >=80) AND (list1[index] <=100) then
        count = count + 1
    endif
next index
print ("Number of integers in range 80-100", count)

for index = 0 to (len(list1) - 1)
    if (list1[index] >=80) and (list1[index] <=100) then
        item = list1[index]
        list1.remove(item)
    endif
next index
print(list1)
```

When the program is coded and run, the first part works correctly but it crashes in the second FOR loop with the message

"if (list1[index] >=80) & (list1[index] <=100):

IndexError: list index out of range

Why does it crash?

It crashes because the length of the list is reduced by one each time an item is removed.

One possible solution is to write the items to a new list if they are not in the range 80-100.

Correct the pseudocode.

```
list2 = [ ]
for index = 0 to (len(list1) - 1)
    if (list1[index] < 80) OR (list1[index] > 100) then
        item = list1[index]
        list2.append (item)
    endif
next index
list1 = list2
print(list1)
```

(See Python program countNumbers 80-100.py in Topic 3 Python programs folder)



3. A program is to be written which merges the following two sorted lists **list1** and **list2** into a single sorted list called **mergeList** and prints out all three lists.

```
list1 = [2,5,15,36,47,56,59,78,156,244,268]
```

```
list2 = [18,39,42,43,66,69,100]
```

- (a) Which list functions will be useful in this program?

`len(list)`, `list.append`

- (b) Write an algorithm to do this in ordinary English. You may find it useful to write the numbers from each list on pieces of paper and do the task manually, or use the bus cards from the previous lesson, split into two sorted lists of uneven length..

Compare next item from list1 with next item from list2

Write the smaller item to mergeList

Repeat, taking next item from the list that had the smaller item

When one list has no more items, move all the remaining items from the other list to mergeList

- (c) Convert the algorithm into pseudocode.

Initialise lists. MergeList is an empty list

Initialise indices i, j for each list to 0

while ((i < len(list1)) and (j < len(list2)))

 if list1[i] < list2[j] then

 mergeList.append (list1[i]) *#append item from list1*

 i = i + 1

 else

 mergeList.append (list2[j]) *#append item from list2*

 j = j + 1

 endif

endwhile

#append any items left in the other list

while i < len(list1)

 mergeList.append (list1[i])

 i = i + 1

endwhile

while j < len(list2)

 mergeList.append (list2[j])

 j = j + 1

endwhile

print (list1, list2, mergeList)

- (d) Code and test the program in a programming language of your choice.

(See Python program MergeLists.py in Topic 3 Python programs folder)



Task 3

4. A linked list abstract data type (ADT) has the following operations:

- create linked list
- add item to linked list
- remove item from linked list

Each node in the linked list consists of a name and a pointer to the next item in the linked list. Items are maintained in alphabetical order.

A variable called start holds the index of the first item in the list

(a) Show the state of the list after the following operations are carried out.

CreateLinkedList

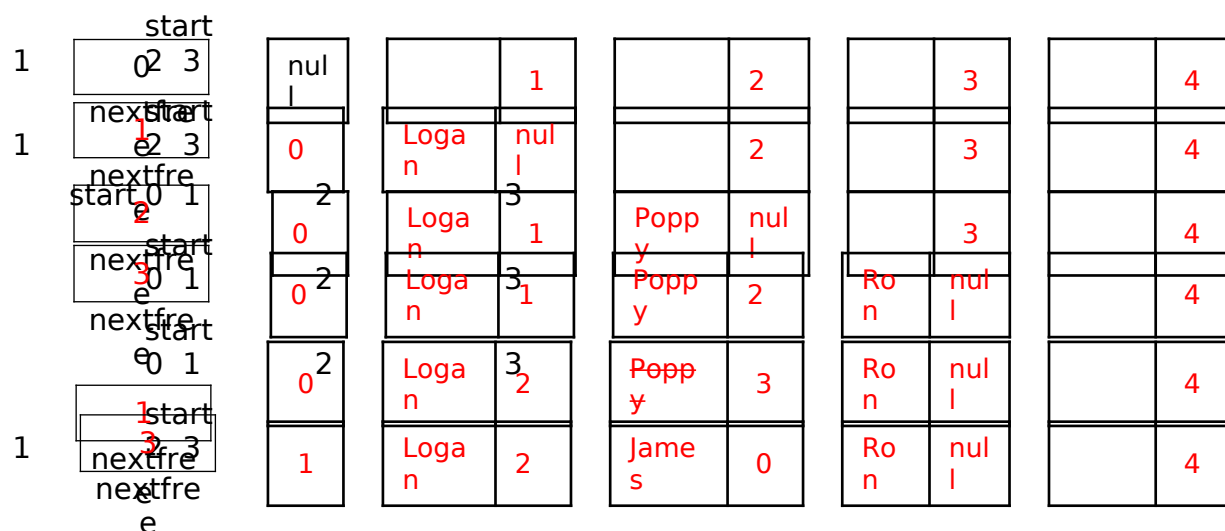
AddItem("Logan")

AddItem("Poppy")

AddItem("Ron")

DeleteItem("Poppy")

AddItem("James")





(b) The linked list is to be implemented as an array of 50 records called myList.

A node is defined as follows:

```
type nodeType
    string name
    integer pointer
endType
```

```
dim myList[0..49] of nodeType
```

The variable pointer holds the index of the next node. A variable called nextfree holds the index of the next free space in the array. The data in the linked list can be accessed in sequence by following the pointers to the next node.

The array is initialised using the following algorithm:

```
for index = 0 to 48
    myList[index].pointer = index + 1
next index
myList[49].pointer = null
start = null
nextfree = 0
```

Show the state of the linked list using the first diagram below, after initialisation of the array.

start = null

nextfree = 0

index	name	pointer
0		1
1		2
2		3
3		4
4		5
:		:
49		null

start = 0

nextfree = 2

index	name	pointer
0	Logan	1
1	Poppy	null
2		3
3		4
4		5
:		:
49		null

(c) Using the second diagram, show the state of the list after the following operations are carried out.

```
CreateLinkedList
AddItem("Logan")
AddItem("Poppy")
```

(d) Refer to the pseudocode on the next page.

(i) Fill in lines 3 and 4 to check for full list



(ii) What is the function of lines 7 - 11?

Handles special case of inserting into an empty list

The procedure AddItem(newItem) is shown below.

```
01 procedure AddItem(newItem)
02 // check if list is full and if so, print error message
03   if nextfree = null then
04     print("List full")
05   else
06     myList[nextfree].name = newName
07     if start = null then
08       temp = myList[nextfree].pointer      //save pointer
09       myList[nextfree].pointer = null
10       start = nextfree
11       nextfree = temp
12   else
13     p = start
14     if newName < myList[p].name then
15       myList[nextfree].pointer = start
16       start = nextfree
17     else
18       placeFound = false                // general case
19       while myList[p].pointer <> null and placeFound = false
20         //peek ahead
21         if newName >= myList[myList[p].pointer].name then
22           p = myList[p].pointer
23         else
24           placefound = True
25       endif
26     endwhile
27     temp = nextFree
28     nextfree = node[nextfree].pointer
29     node[temp].pointer = node[p].pointer
30     node[p].pointer = temp
31   endif
32 endif
```



33 endif

34 endprocedure

(iii) What condition is line 14 of the pseudocode checking for?

Special case of inserting at the front of the list

(iv) Show the state of the list after three further operations:

AddItem("Alan")

DeleteItem("Poppy")

AddItem("James")

start = 2

nextfree

index	name	pointer
0	Logan	null
1	James	0
2	Alan	1
3		4
4		5
:		:
49		null



5. Deleting an item from a linked list.
Here is an alphabetically ordered linked list, ListA, of animals. This implementation uses:
- a variable **start** to indicate the first item in the list
 - a null in the pointer field to indicate the end of the list

index	animal	pointer
0	Snake	null
1	Dog	2
2	Mouse	0
3	Ant	1
4		5
5		Null

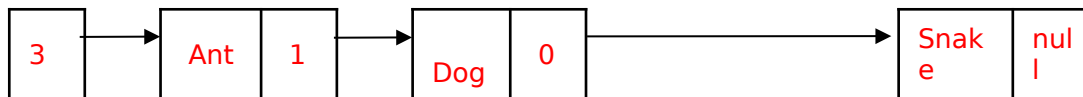
start = 3

nextfree
= 4

- (a) (i) What is the value of `listA[start].pointer`? **1**
- (ii) What is the value of `listA[listA[start].pointer].pointer`? **2**
- (iii) If `p = 1`, what is the value of `listA[listA[p].pointer].name`? **Mouse**
- (b) The following pseudocode deletes an item in the table.
- ```
01 xName = "Mouse"
02 // check for empty list
03 if start = null then
04 print ("List is empty")
05 else
06 p = start
07 if deleteName = listA[start].name then
08 start = listA[start].pointer
09 else
10 while deleteName <> listA[listA[p].pointer].name
11 p = listA[p].pointer
12 endwhile
13 endif
14 endif
15 nextptr = listA[p].pointer
16 listA[p].pointer = listA[nextptr].pointer
```



- (i) Complete the diagram below to show the list after deleting Mouse according to the algorithm given in the pseudocode.



- (ii) Complete the table below after deleting Mouse
- (iii) What special case is line 7 of the pseudocode checking for?

The node to delete is the head of the list.

| index | animal | pointer |
|-------|--------|---------|
| 0     | Snake  | null    |
| 1     | Dog    | 0       |
| 2     | Mouse  | 0       |
| 3     | Ant    | 1       |
| 4     |        | 5       |
| 5     |        | Null    |

start = 3

nextfree

- (iv) In the pseudocode given, the space left by the deleted item is not linked back into the list of free space. Explain how this could be done.

Set the pointer in deleted node equal to nextfree  
 Set nextfree = index of deleted node (i.e.2)

Show below what each node would hold if this was done.

| index | animal | pointer |
|-------|--------|---------|
| 0     | Snake  | null    |
| 1     | Dog    | 0       |
| 2     | Mouse  | 4       |
| 3     | Ant    | 1       |
| 4     |        | 5       |
| 5     |        | Null    |

start = 3

nextfree